Oracle9*i* Application Developer's Guide - XML
Release 1 (9.0.1)
Part Number A88894-01

11 Using JDeveloper to Build Oracle XML Applications

This chapter contains the following sections:

Introducing JDeveloper9i

Oracle JDeveloper9i is a J2EE™ development environment with end-to-end support for developing, debugging, and deploying e-business applications. JDeveloper empowers users with highly productive tools, such as the industry's fastest Java debugger, a new profiler, and the innovative CodeCoach tool for code performance analysis and improvement.

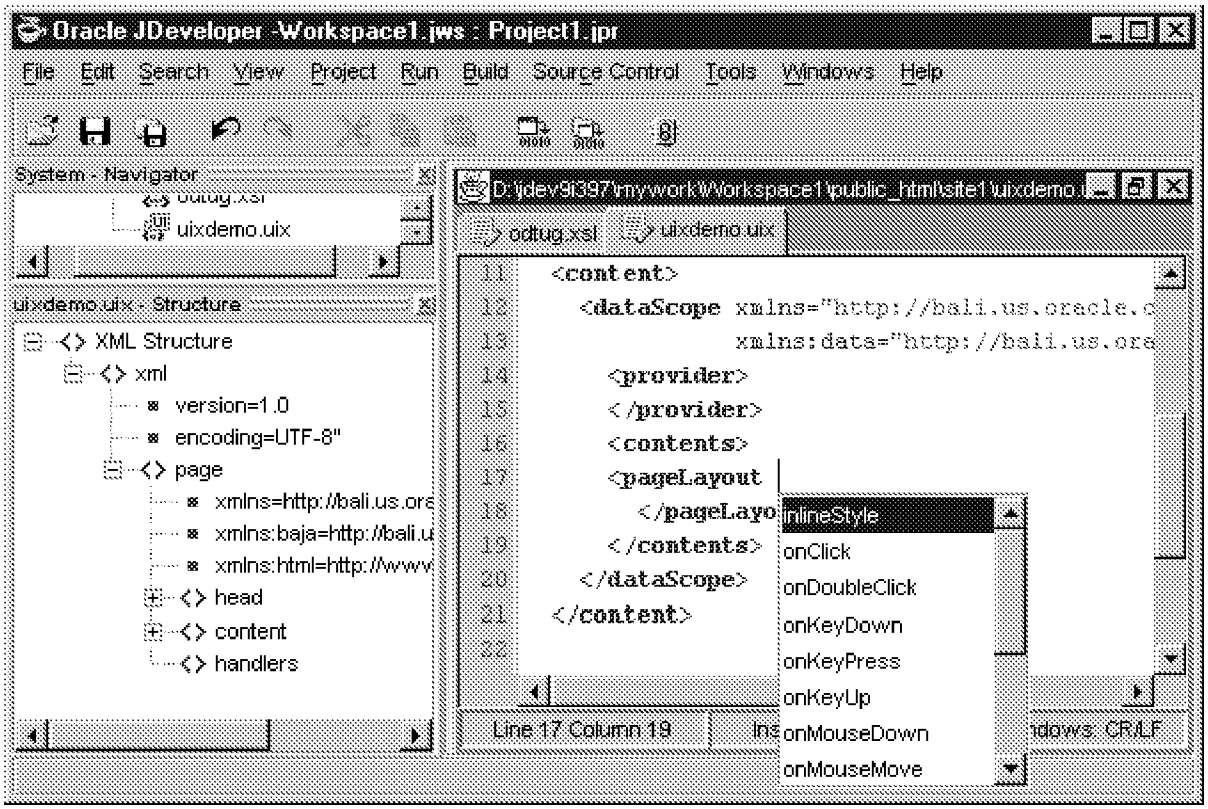
To take J2EE application development to a higher level of productivity, JDeveloper now offers Business Components for Java (BC4J), a standards-based, server-side framework for creating scalable, high-performance Internet applications. The framework provides design-time facilities and runtime services to drastically simplify the task of building and reusing business logic.

JDeveloper9i has a new schema-driven XML editor. See Figure 11-1. An XML Schema Definition defines the structure of an XML document and is used in the editor to validate the XML and help developers when typing. This feature is called *Code Insight* and provides a list of valid alternatives for XML elements or attributes in the document. Just by specifying the schema for a certain language, the editor can assist you in creating a document in that markup language.

Oracle JDeveloper9i simplifies the task of working with Java application code and XML data and documents at the same time. It features drag-and-drop XML development modules. These include the following:

- Color-coded syntax highlighting for XML
- Built-in syntax checking for XML and Extensible Style Sheet Language (XSL)
- XSQL Pages and Servlet support, where developers can edit and debug Oracle XSQL Pages, Java programs that can query the database and return formatted XML, or insert XML into the database without writing code. The integrated servlet engine allows you to view XML output generated by Java code in the same environment as your program source, making it easy to do rapid, iterative development and testing.
- Includes Oracle's XML Parser for Java
- Includes XSLT Processor
- Related XDK for JavaBeans components
- XSQL Page Wizard. See "Page Selector Wizard" .
- XSQL Element Wizard. See "XSQL Element Wizard" .
- XSQL ActionHandlers
- Schema-driven XML editor.

Figure 11-1 JDeveloper9i's Schema-Driven XML Editor in Action



Text description of the illustration jdevedit.gif

Oracle XML Developers Kit (XDK) is integrated into JDeveloper, so that Jdeveloper offers many utilities that help Java developers handle, create, and transform XML. For example, when designing with XSQL Servlet, you can query and manipulate database information, generate XML documents, transform them using XSLT stylesheets, and make them available on the web.

See Also:

Chapter 10, "XSQL Pages Publishing Framework"

Business Components for Java (BC4J)

Oracle Business Components for Java is a 100%-Java, XML-powered framework that enables productive development, portable deployment, and flexible customization of multi-tier, database-savvy applications from reusable business components.

Application developers use the Oracle Business Components framework and Oracle JDeveloper 's integrated design-time wizards, component editors, and productive Java coding environment to assemble and test application services from reusable business components.

These application services can then be deployed as either CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

The same server-side business component can be deployed without modification as either a JavaServer Pages/Servlet application or Enterprise JavaBeans component. This deployment flexibility, enables developers to reuse the same business logic and data models to deliver applications to a variety of

clients, browsers, and wireless Internet devices without having to rewrite code.

In JDeveloper, you can customize the functionality of existing Business Components by using the new visual wizards to modify your XML metadata descriptions.

Oracle JDeveloper XML Strategy

Oracle JDeveloper9i supports building XML applications. JDeveloper new integrated schema-driven XML code editor works on XML Schema-based documents such as:

- For creating XML Schemas
- For creating XSLT Stylesheets,...

with "tag-insight" to help you easily enter the correct elements and attributes as defined by the schema. In addition to the editing capabilities, JDeveloper's XML code editor also has the following features:

- Error highlighting
- Property inspection
- Tree-like view in the structure pane

Further Information

See Also:

- http://otn.oracle.com/products/jdev/
- *Oracle9i Java Developer's Guide*
- *Oracle JavaServer Pages Developer's Guide and Reference*
- *Oracle9i CORBA Developer's Guide and Reference*
- *Oracle9i Enterprise JavaBeans Developer's Guide and Reference*
- *Oracle9i Java Stored Procedures Developer's Guide*
- *Oracle9i Java Tools Reference*
- *Oracle9i JDBC Developer's Guide and Reference*
- *Oracle9i JPublisher User's Guide*
- *Oracle9i Oracle Servlet Engine User's Guide*
- *Oracle9i SQLJ Developer's Guide and Reference*

What's Needed to Run JDeveloper9i

JDeveloper9i is an IDE that has been written in Java and therefore, runs on Windows NT, Windows 2000, Linux and Solaris operating systems. It needs a minimum of 128 Mb RAM.

Minimum system requirements for JDeveloper

Refer to JDeveloper Release Notes. As more products are run on the same machine, system requirements are increased. A typical development environment for running JDeveloper includes:

- Running JDeveloper
- Running Oracle9*i* locally
- Running Oracle9i Application Server locally
- Additional third party tools (profilers, version control, modelers,...)

These add to system requirements, in terms of actual CPU usage and in disk space needs.

Accessing JDeveloper9i

The beta release of JDeveloper9i will be available in the summer of 2001 from the Oracle Technology Network (OTN) at http://otn.oracle.com.

XML in Business Components for Java (BC4J)

The Business Components for Java (BC4J) framework in JDeveloper9i uses XML to define the metadata that represents the declarative settings and features of the objects. Custom or complex business logic can be implemented in Java.

- The BC4J Tester enables you to see data in view objects as XML.
- Business rules, such as validation rules, are stored in XML rather than Java source code
- Easy customization of business applications by changing XML rather than Java source code
- Applications are easier to read and understand by abstracting the logic in XML

BC4J uses XML to Store Metadata

The business components for Java framework that ships with JDeveloper uses XML to store metadata about its application components. Important information is now stored in a structured document rather than in Java source code. This makes the application easier to understand and customize.

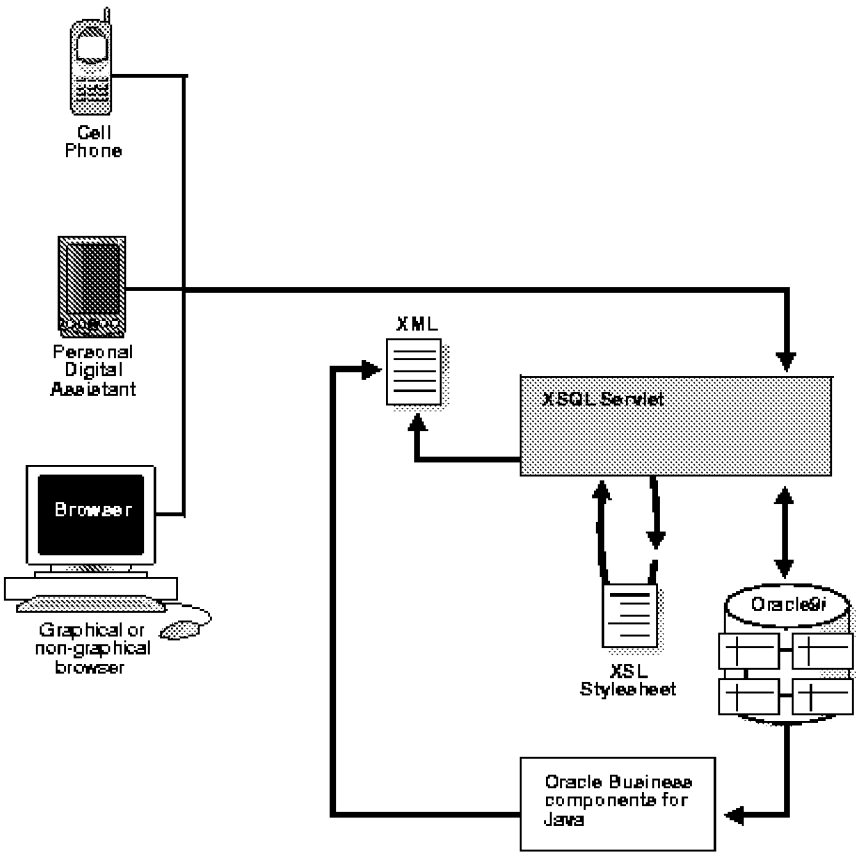
The application is now customizable without having access to the source code.

Figure 11-2 shows how BC4J is used with XSQL servlet to generate XML documents.

See Also:

http://otn.oracle.com/products/bc4j/

Figure 11-2 Using Business Components for Java (BC4J)



Text description of the illustration adxml086.gif

Business rules can be changed on site without needing access to the underlying component source code.

Building XSQL Clients with Business Components for Java (BC4J)

In JDeveloper 9i, you can build XSQL Pages which can integrate with BC4J application modules and thereby serve application logic from the middle tier to multiple clients. You can retrieve XML data and present it to any kind of a client device just by applying the corresponding stylesheet.

The following features will assist you in building XSQL clients with BC4J:

- Object Gallery

- XSQL Element Wizard
- Page Selector Wizard

Note:

These appearance of these features may differ in the JDeveloper9i production version.

See Also:

Chapter 12, "Building BC4J and XML Applications"

Object Gallery

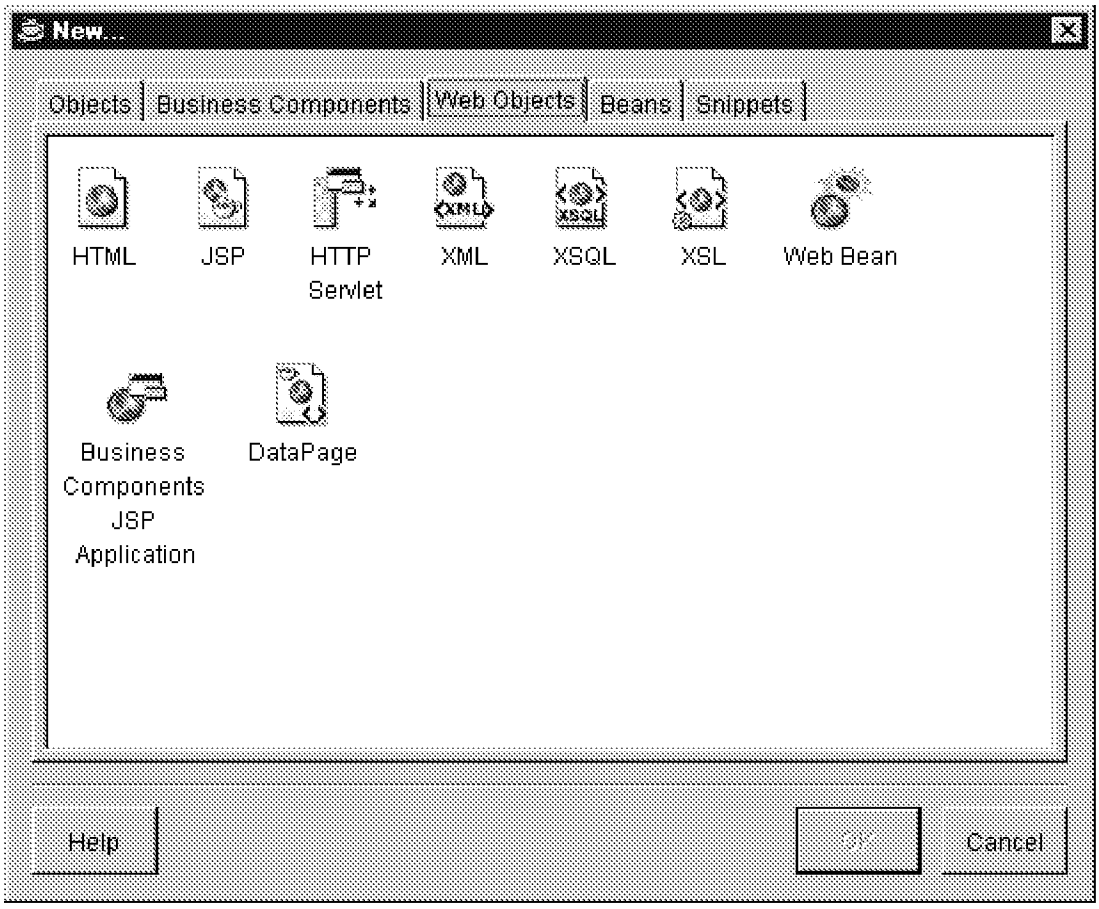
The Web Object Gallery has icons to assist in creating XSQL, XML, and XSL documents easily. When you click on them, the basic tags for these pages are generated and you can then enhance them.

The XSQL Pages icon is of special interest because the XSQL Element Wizard can be used, after generating your basic XSQL pages, to insert data bound tags in the XSQL pages. Figure 11-3 illustrates JDeveloper's Object Gallery.

See Also:

"XSQL Element Wizard" .

Figure 11-3 JDeveloper's Object Gallery Showing the new XSQL, XML, and XSL Icons



Text description of the illustration jdevxml.gif

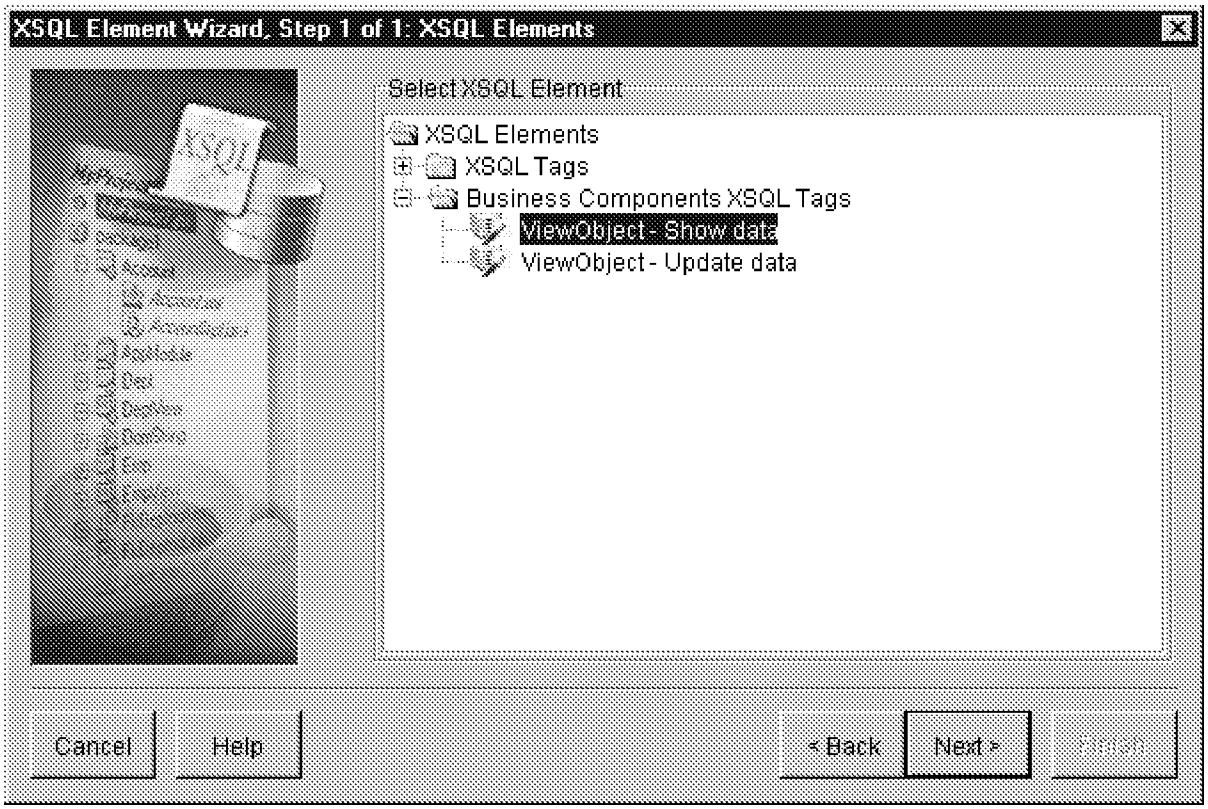
Note:

The appearance of these features (wizards) may change in the production release.

XSQL Element Wizard

XSQL Element Wizard provides you with a mechanism to add tags which allows accessing database tables or BC4J View Objects. You can either perform queries against them or update the underlying database tables through them. Figure 11-4 illustrates the JDeveloper9i XSQL Element Wizard.

Figure 11-4 JDeveloper's XSQL Element Wizard



Text description of the illustration jdevxsql.gif

Note:

The appearance of these features (wizards) may change in the production release.

Page Selector Wizard

When you need create XSQL pages while building a web application, you can invoke Page Wizard which allows you to create XSQL Pages on top of either database tables directly or on top of BC4J View Objects. When you choose to build an XSQL Page on top of a BC4J View Object, you are prompted to select an application module from a list or create a new application module and then build the XSQL Pages based application.

See Also:

Oracle9i Java Developer's Guide

XML Features in JDeveloper9i

The following lists JDeveloper9i's supported Oracle XML Developer's Kit for Java (XDK for Java) components:

- Oracle XML Parser for Java

- Oracle XSQL Servlet

You can use the XML Parser for Java including the XSLT Processor and the XML SQL Utility in JDeveloper as all these tools are written in Java. JDeveloper provides these components.

Sample programs which demonstrate how to use these tools can be found in the `[JDeveloper]/Samples/xmlsamples` directory.

Oracle XDK and Transviewer Beans Integration

Oracle XDK for Java consists of the following XML tools:

- XML Parser for Java
- XML- SQL Utility for Java
- XML Java Class Generator
- XSQL Servlet
- XML Transviewer Beans

All these utilities are written in Java and hence can easily be dropped into JDeveloper and used 'out of box'. You can also update the XDK for Java components with the latest versions downloaded from Oracle Technology Network (OTN) at http://technet.oracle.com/tech/xml.

Oracle XDK for Java also includes the XML Transviewer Beans. These are a set of Java Beans that permit the easy addition of graphical or visual interfaces to XML applications. Bean encapsulation includes documentation and descriptors that make them accessible directly from JDeveloper. You can drop these beans into the TOOLS palette and use them to build applications such as XML/XSL editors.

> See Also:
>
> Chapter 23, "Using XML Transviewer Beans" for more information on how to use the Transviewer Beans.

Oracle XML Parser for Java

Including the Oracle XML Parser for Java in your project allows you to write applications that can search and process XML documents. You can include the Oracle XML Parser in your project with one click as JDeveloper has a built-in library for it.

Code Insight makes understanding and using the code easier and in-place access to JavaDoc on the classes for reference. The XML parser for Java facilitates processing an XML document using either of the following interfaces:

- DOM: a tree of W3C DOM
- SAX: a stream of SAX events

Oracle XSQL Servlet

The XSQL Servlet is a tool that processes SQL queries and outputs the result set as XML. This processor is implemented as a Java servlet and takes as its input an XML file containing embedded SQL queries. It uses the XML Parser for Java and the XML SQL Utility to perform many of its operations.

The XSQL Servlet offers a productive and easy way to get XML in and out of the database. Using simple scripts you can:

- Generate simple and complex XML documents
- Apply XSL Stylesheets to generate into any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

JDeveloper XSQL Example 1: emp.xsql

For example, consider the following XML example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<FAQ xmlns:xsql="urn:oracle-xsql" connection = "scott">
  <xsql:query doc-element="EMPLOYEES" row-element="EMP">
    select e.ename, e.sal, d.dname as department
    from dept d, emp e
    where d.deptno = e.deptno
  </xsql:query>
</FAQ>
```

Generates the following:

```
<EMPLOYEES>
  <EMP>
    <ENAME>Scott</ENAME>
    <SAL>1000</SAL>
    <DEPARTMENT>Boston</DEPARTMENT>
  </EMP>
  <EMP>
  ...
</EMPLOYEES>
```

With JDeveloper9i you can easily develop and execute XSQL files. The built in Web Server and the user's default Web Browser will be used to display the resulting pages.

Using ActionHandlers in XSQL Pages

XSQL ActionHandlers are Java classes which can be invoked from XSQL Page applications very easily. Since these are Java classes they can be debugged from JDeveloper just like any other Java application.

If you are building an XSQL Pages application, you can make use of the XSQL Action Handler to extend the set of actions that can be performed to handle more complex jobs. You will need to debug this ActionHandler.

Your XSQL Pages should be in the directory specified in the Project Property "HTML Paths" settings for "HTML Source Directory".

To debug your ActionHandler carry out these steps:

1. Assume you have created an .xsql file which has reference to a custom ActionHandler called MyActionHandler.

2. Debug this ActionHandler because it is not exactly behaving as you expect.

3. Set breakpoints in your Java source file.

4. He right mouse clicks on the .xsql file and now chooses Debug... from the menu.

See Also:

The JDeveloper Guide under the online HELP menu.

XML Data Generator Web Bean

Oracle JDeveloper has an XML Data Generator Web Bean. It generates XML containing the data from a View Object and renders it to the output stream of a JSP response.

You can author JSP pages that use XML and XSL to render a response to the client.

This XML Web Bean can be used in JSP and Servlet applications. It reads data from a Business Component (View Object) and produces the appropriate XML. The strength of this Web Bean is that it analyzes the Business Component Application and navigates through it's hierarchy to produce the nested XML.

The XML Web Beans also allows the specification of an XSL Stylesheet. In addition to XML, the Web Bean can then generate HTML, WML, transformed XML and any other text format.

Mobile Application Development with Portal-To-Go and JDeveloper

Portal-To- Go and Oracle JDeveloper together offer an extremely powerful environment for developing mobile applications. Developers can use JDeveloper to generate XML from the database or from a Business Components for Java Application and use Portal-To-Go to deliver content to Web browsers, PDAs, or Cell phones.

Building XML Applications with JDeveloper

Consider the following example that demonstrates how XML is used to represent data, not present it. It shows the many to one relationship between employees and departments.

JDeveloper XML Example 1: BC4J Metadata

```
<Departments>
<Dept>
   <Deptno>10</Deptno>
   <Dname>Sales></Dname>
   <Loc>
   <Employees>
     <Employee>
        <Empno>1001></Empno>
        <Ename>Scott</Ename>
        <Salary>80000</Salary>
     </Employee>
   </Employees>
...
     </Employee>
   </Employees>
</Dept>
<Dept>
...
```

Procedure for Building Applications in JDeveloper9i

To build this project in JDeveloper9i carry out the following steps:

1. Start a New JDeveloper Project by selecting File > New Project.
2. Create a Business Components for Java application.
3. Create an XSQL Page based upon a BC4J application module, by invoking the Page Selector Wizard.
4. Select the application module from the list that pops up.
5. Select the View Object on which you want to base your XSQL Page.
6. Select the columns that you want to view.

When you finish these steps in the Page Wizard, you should have an XSQL Page based on the Business Components for Java (BC4J) framework View objects. When you run this page, it sends the XML data to your browser.

You could optionally create a stylesheet to format the data so that it appears in a way that you prefer or you can tune it so that it can be displayed on a PDA or cellphone.

Using JDeveloper's XML Data Generator Web Bean

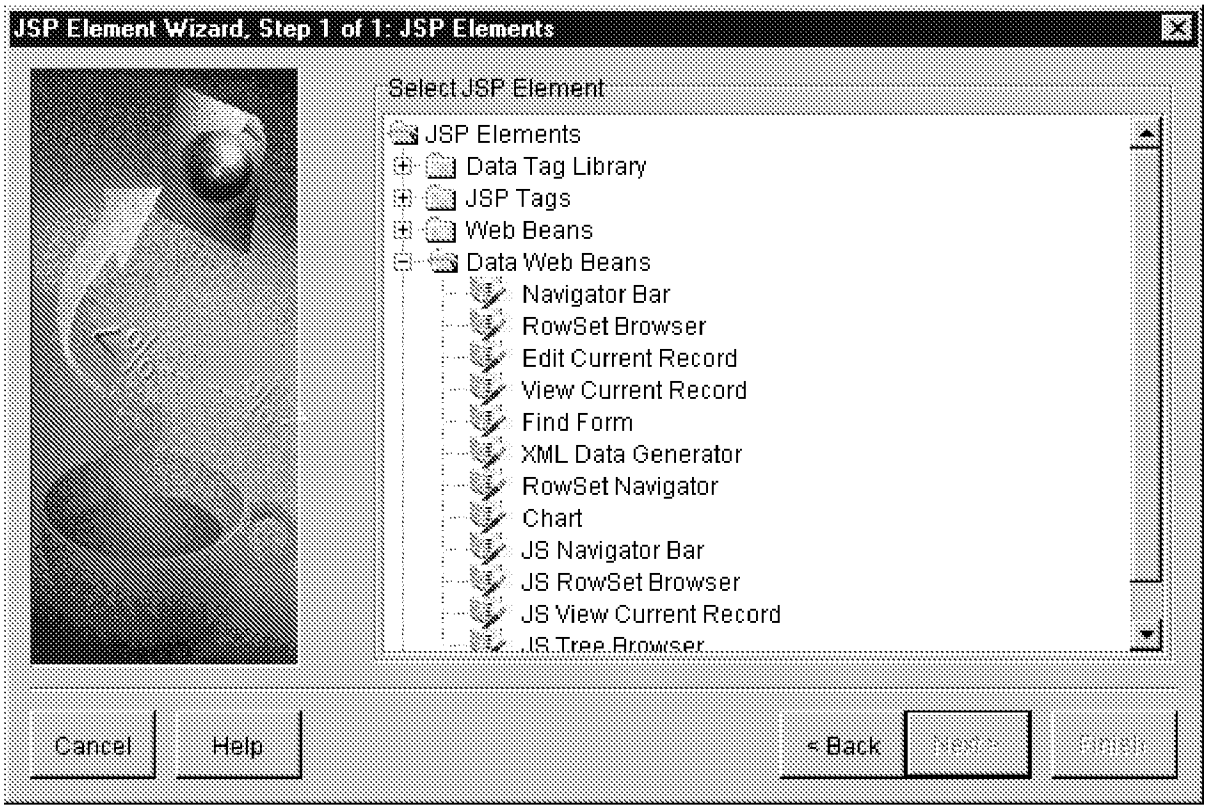
The XML Data Generator Web (Bean) can be used in JSP and Servlet applications. It reads data from a Business Component (View Object) and produces the appropriate XML. The strength of this Web Bean is the following:

- It analyzes the Business Component Application and navigates through it's hierarchy to produce the nested XML.
- It allows specification of an XSL Stylesheet. The Web Bean can then generate HTML, WML, transformed XML, and any other text format.

The Data Generator Web Bean is in the "Data Web Beans" category of the JSP Elements wizard. Figure 11-5 illustrates accessing the XML Data Generator Web (Bean) from the JSP Element Wizard.

Call the Element Wizard from your JSP or XSQL Page by right-clicking anywhere on the Page where you want to include an element. Specify the stylesheet as a parameter in the Element Wizard.

Figure 11-5 JSP Element Wizard: XML Data Generator Bean



Text description of the illustration jdevjsp.gif

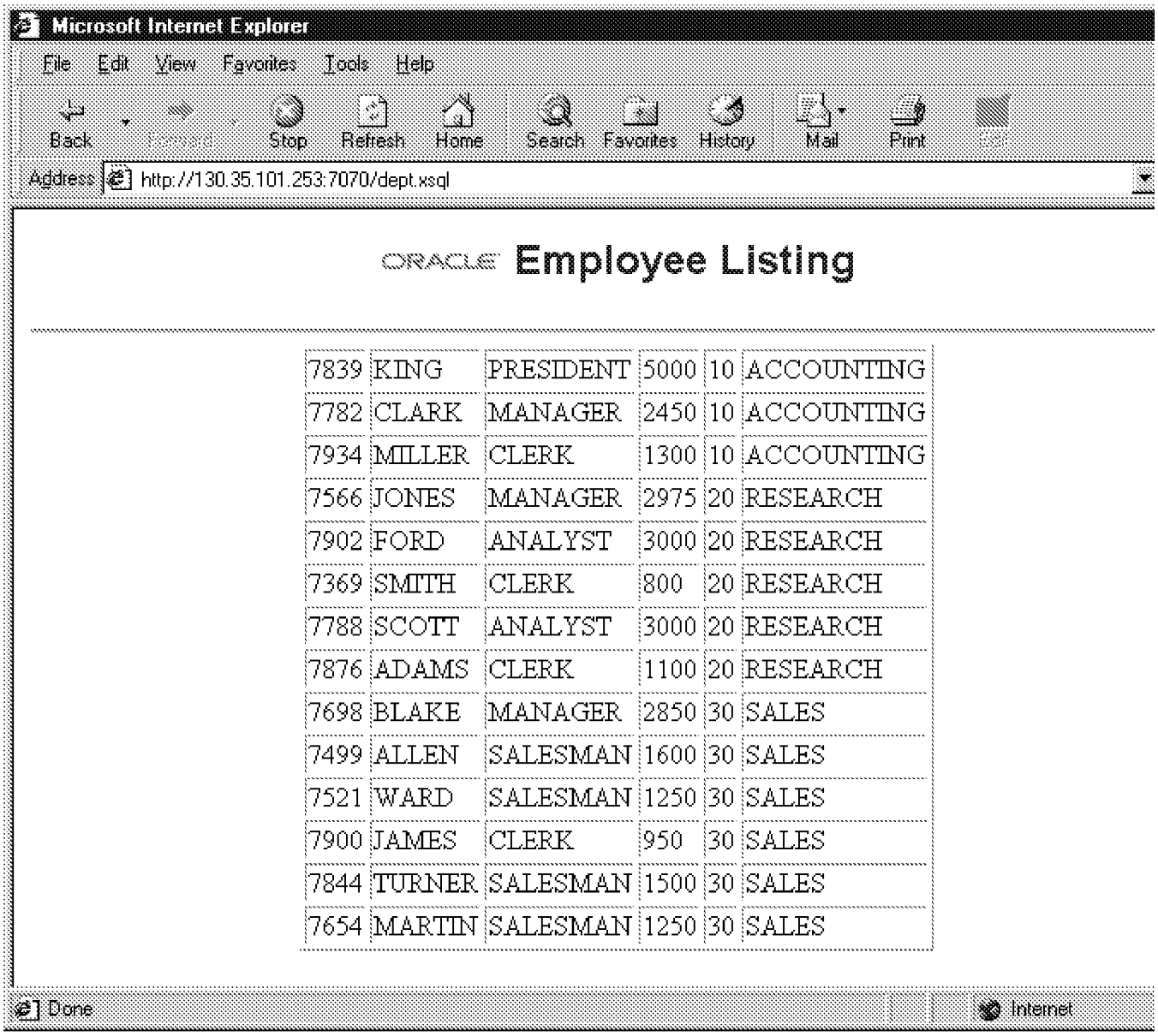
Note:

The appearance of these features (wizards) may change in the production release.

Once you launch this wizard, you can specify a stylesheet to apply to the XML data that you generate and see the result of your output.

Figure 11-6 is an example output displayed by applying an XSL stylesheet to the employee listing.

Figure 11-6 Browser HTML Display Showing the Employee Listing (XML+XSLT= HTML)



Text description of the illustration jdev_07.gif

Using XSQL Servlet from JDeveloper

XSQL Servlet offers a productive and easy way to get XML in and out of the database.

See Also:

Chapter 3, "Oracle XML Developer Kits (XDKs) and Components: Overview and General FAQs" and Chapter 10, "XSQL Pages Publishing Framework" for information about how to use XSQL Servlet.

When using XSQL Servlet in JDeveloper, you do not need to include the XSQL Runtime in your project as this is already done for any new XSQL Page or XSQL wizard-based application.

Using simple scripts you can do the following from JDeveloper:

- Generate simple and complex XML documents
- Apply XSL stylesheets to generate into any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

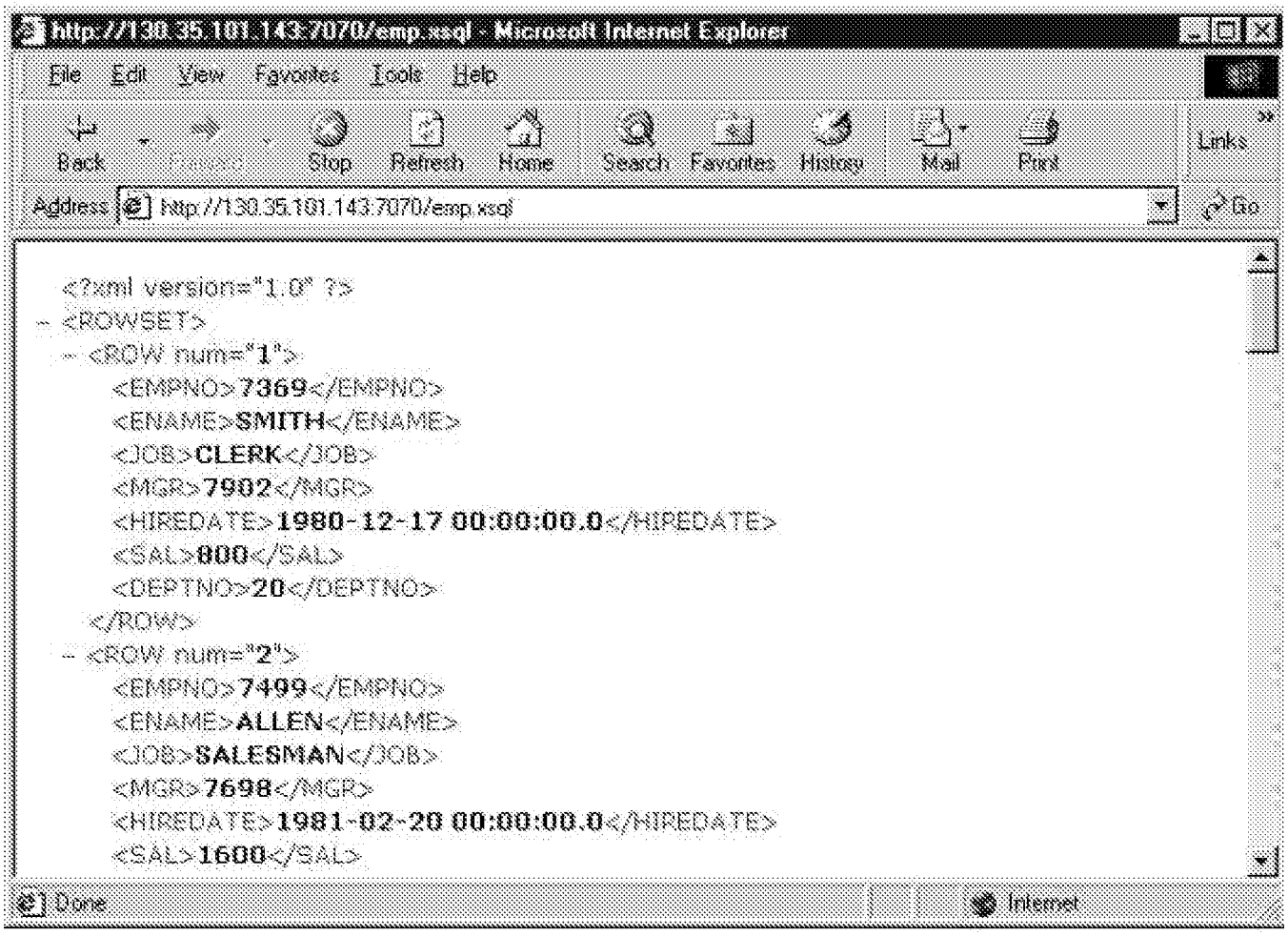
Consider a simple query in an XSQL file, which returns details about all the employees in the emp table. The XSQL code to get this information would be as shown in Example 2:

JDeveloper XSQL Example 2: Employee Data from Table emp: emp.xsql

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    select *
    from emp
    order by empno
</xsql:query>
```

Figure 11-7 shows what the raw employee XML data displayed on the browser.

Figure 11-7 Employee Data in raw XML Format



Text description of the illustration jdev_08.jpg

If you want to output your data in a tabular form as shown in Figure 11-6, make a small modification to your XSQL code to specify a stylesheet. The changes you would make in this example are shown below highlighted.

JDeveloper XSQL Example 3: Employee Data with Stylesheet Added

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
    select *
    from emp
    order by empno
</xsql:query>
```

The result would be like the table shown in Figure 11-6. You can do a lot more with XSQL Servlet of

course.

See Also:

Chapter 10, "XSQL Pages Publishing Framework" and also the XDK for Java, XSQL Servlet Release Notes on OTN at http://technet.oracle.com/tech/xml

Creating a Mobile Application in JDeveloper

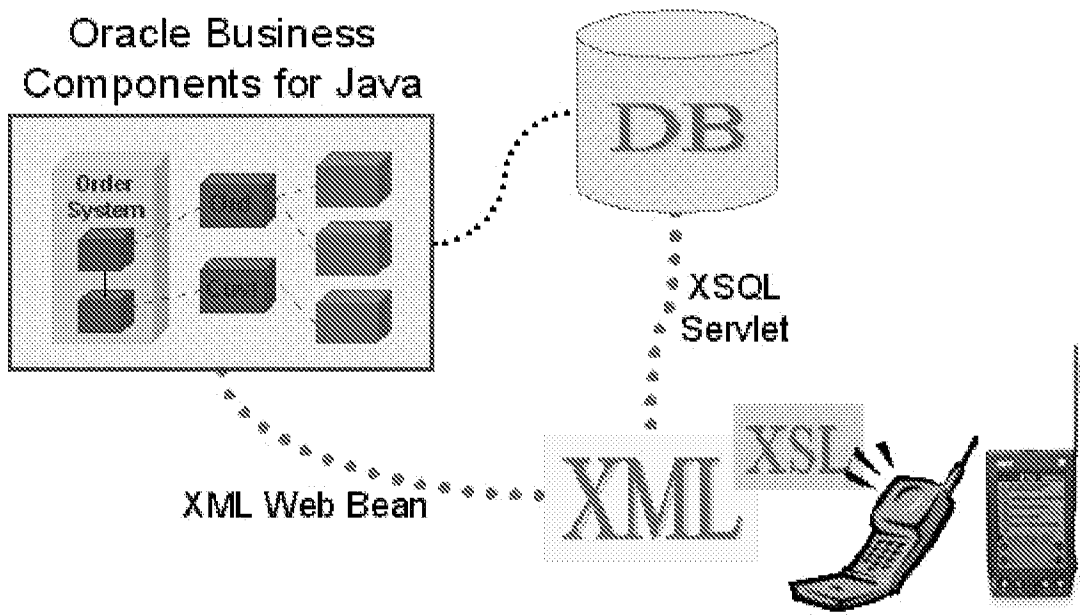
This mobile application is a Departments database application that demonstrates how Business Components for Java (BC4J) and XML can be used to develop applications that can be accessed over wireless devices. The application consists of two main parts:

- Server-side business logic which is developed using the Business Components for Java (BC4J) Framework and the second is the client part. The business logic consists of a view object based on the DEPT table in SCOTT's schema.
- A mechanism to query the DEPT table and update it from any client device including a browser, a cellular phone and a Palm Pilot. For the latter device, the application uses emulators running on Windows NT.

Figure 11-8 shows schematically how the mobile application works with BC4J, XSQL Servlet, XSL Stylesheets, and Oracle9*i*.

You can see a more comprehensive demo of a similar application on http://otn.oracle.com/tech/xml

Figure 11-8 Creating a Mobile Application in JDeveloper Using BC4J and XSQL Servlet

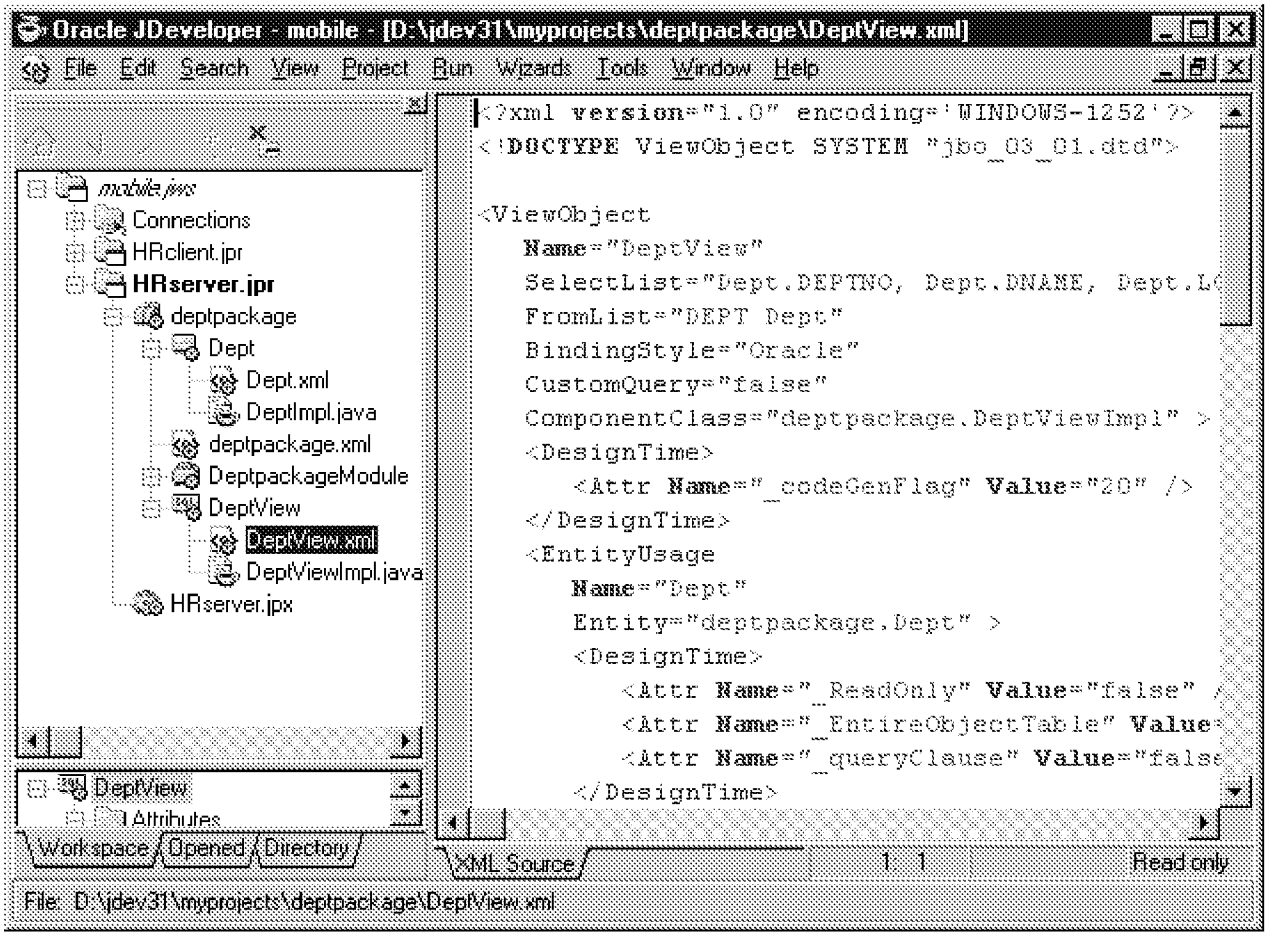


Text description of the illustration jdev_12.jpg

1 Create the BC4J Application

First create the BC4J application. It connects to the SCOTT schema on an Oracle9*i* database. Figure 11-9 shows the XML file containing the metadata about the DEPT object. See "JDeveloper XML Example 1: BC4J Metadata" .

Figure 11-9 BC4J Application: DEPT View Object XML File

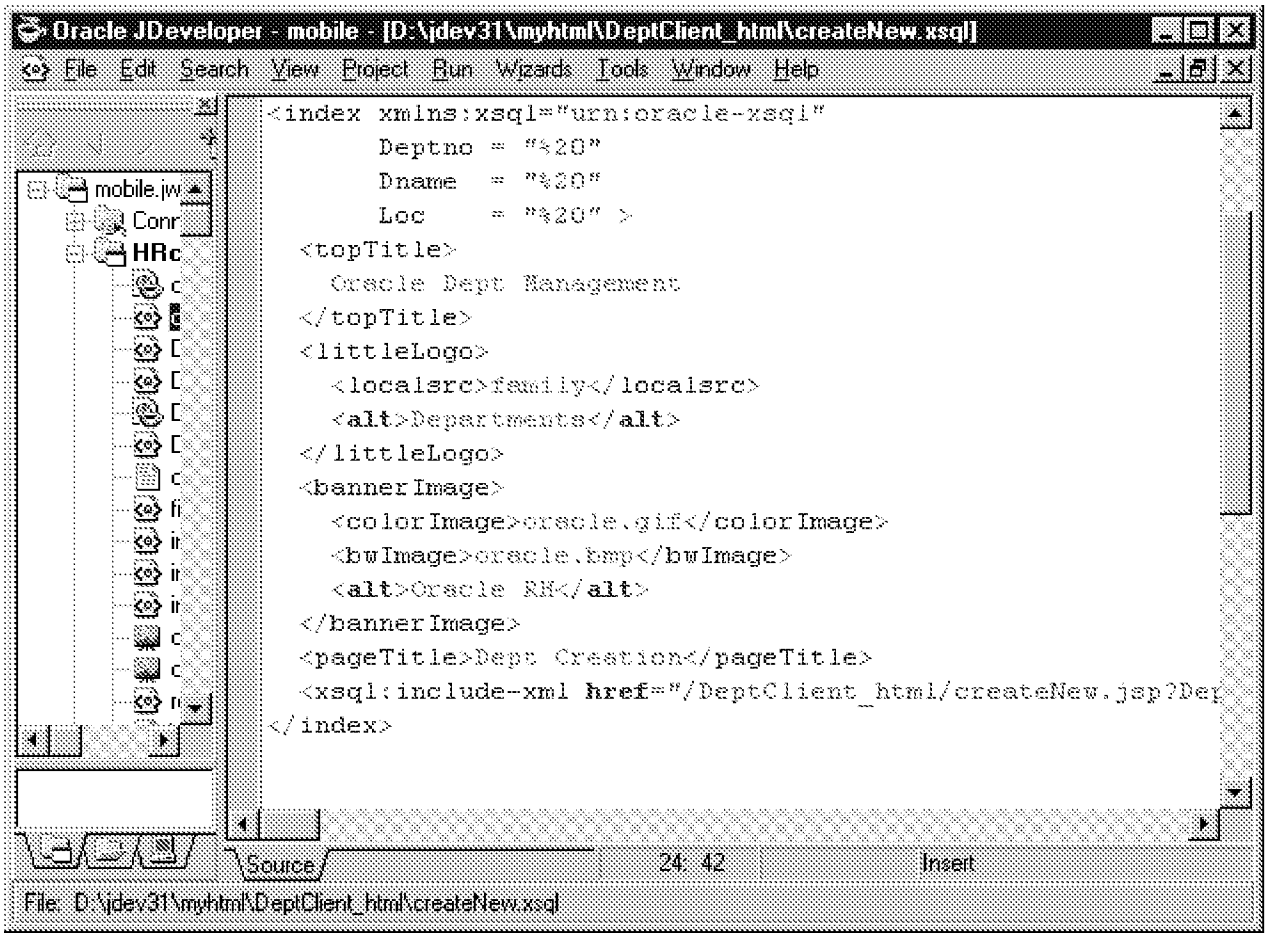


Text description of the illustration jdev13.jpg

2 Create JSP Pages Based on a BC4J Application

You can then create JSP pages based upon this BC4J application. In the JSP pages you are introduced to the XML Data Generator Web Beans. Figure 11-10 shows the XSQL file which calls the JSP page to create the new department.

Figure 11-10 BC4J Application: XSQL File Calling JSP Page



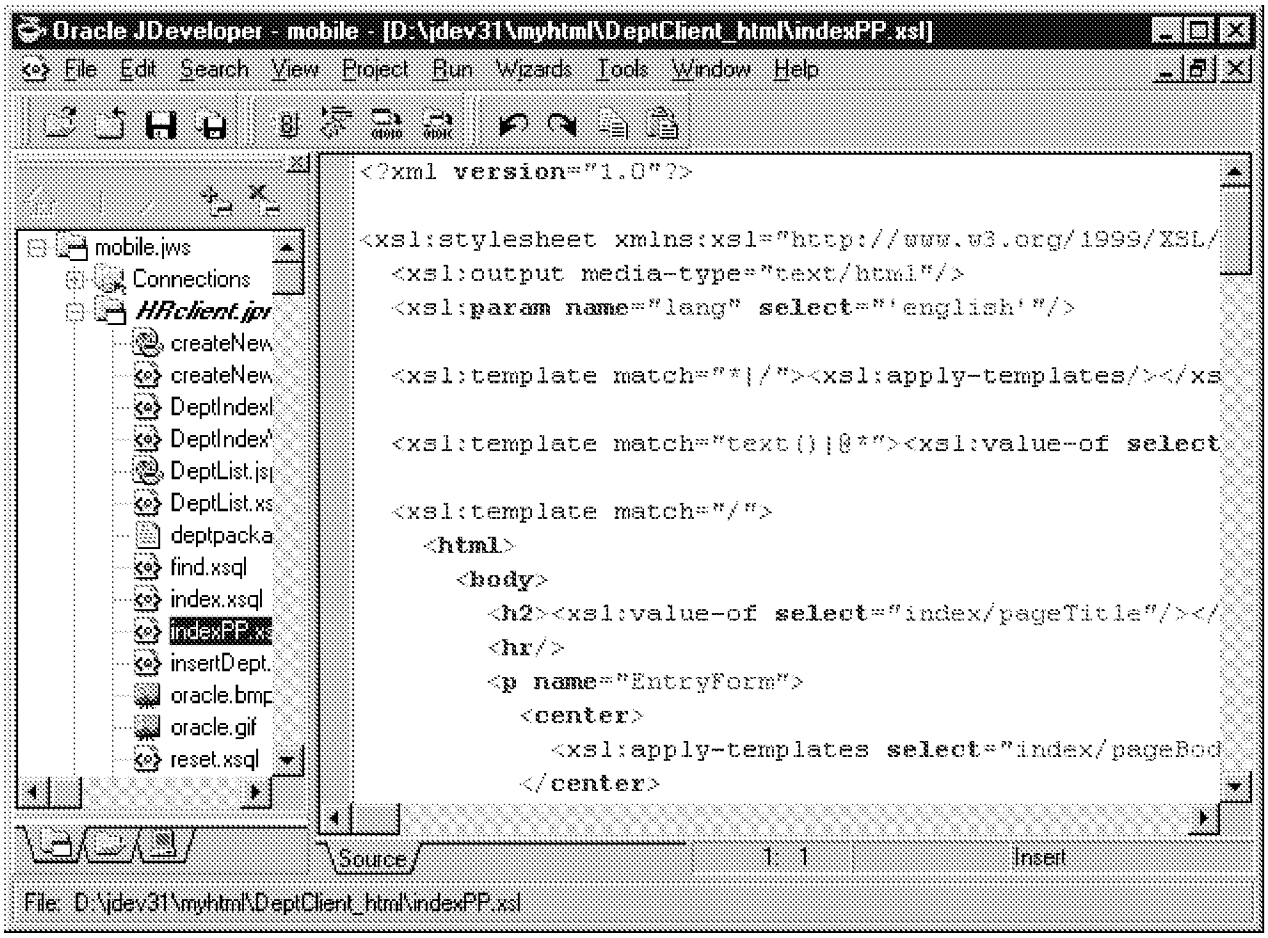
Text description of the illustration jdev14.jpg

3 Create XSLT Stylesheets According to the Devices Needed to Read The Data

We create XSLT stylesheets to go with the various client devices that we are going to access our data from. In your XSQL files, you specify the list of stylesheets and the protocols they go with which basically ties the stylesheets to the client device.

Figure 11-11 shows an example code snippet of a stylesheet (indexPP.xsl) which transforms the XML data to HTML for displaying on a browser on the Palm Pilot emulator.

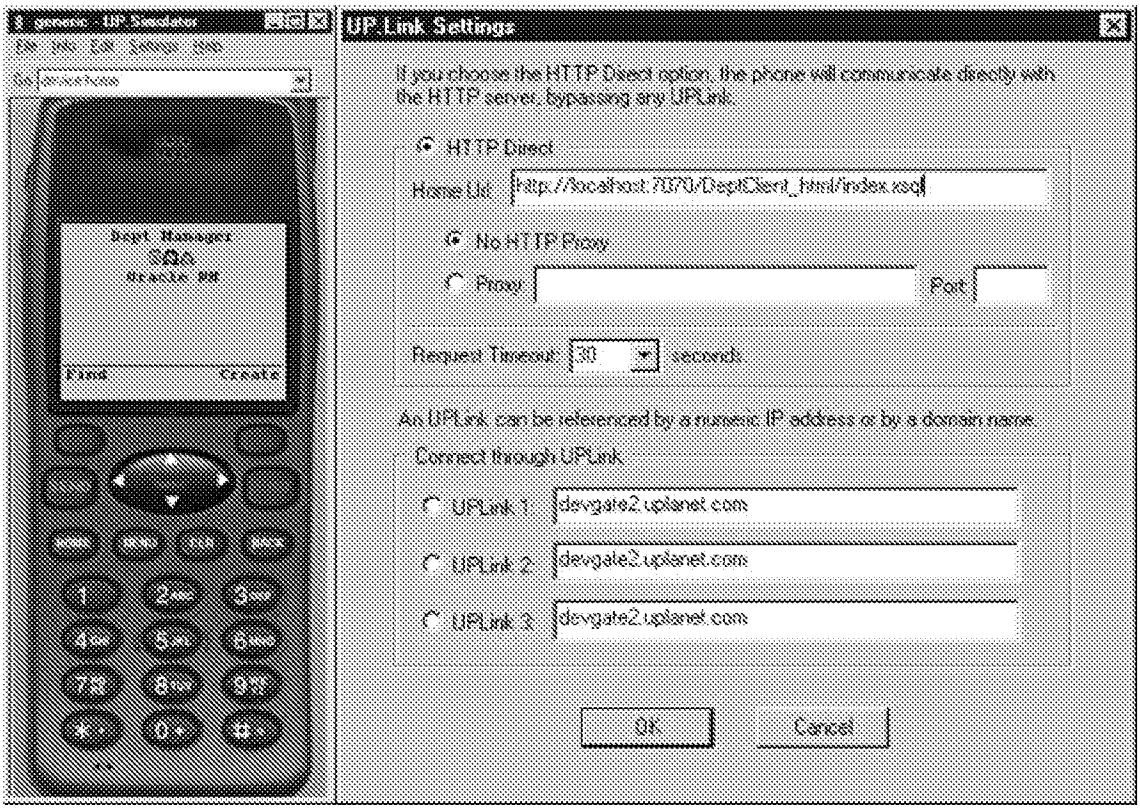
Figure 11-11 BC4J Application: XSL Stylesheet (indexPP.xsl)



Text description of the illustration jdev15.jpg

Figure 11-12 shows the Cell Phone Emulator running the Departments Application Client. It also shows the setup screen for the Cell Phone Emulator.

Figure 11-12 Cell Phone Emulator Running the Department Application Client



Text description of the illustration jdev_10.jpg

Figure 11-13 shows the Palm Pilot Emulator accessing the Departments Application via HandWeb Browser.

Figure 11-13 Palm Pilot Emulator Accessing the BC4J Departments Application Through HandWeb Browser

Text description of the illustration jdev_11.jpg

Frequently Asked Questions (FAQs): Using JDeveloper to Build XML Applications

Constructing an XML Document in JSP

Question

I am dynamically constructing an XML document in a JSP page (from the results of data returned by a PL/SQL API) using the classes generated by the Class generator (based on a DTD) and then applying a XSL stylesheet to render the transformation in HTML. I see that this works fine only for the first time, i.e when the JSP is first accessed (and internally compiled), and fails every time the same page is accessed thereafter.

The error I get is:

```
"oracle.xml.parser.v2.XMLDOMException: Node doesn't belong to the current
document"
```

The only way to make it work again is to compile the JSP, by just 'touching' the JSP page. Of course, this again works only once. I am using Apache JServ.

How can this be overcome? Does the 'static' code in the Java class generated for the top level node have to do anything with it?

Answer

It seems to me that you may have stored some "invalid" state in your JSP. And the XML Parser picks this "invalid" state, then, throws the exception you mentioned.

As far as I know, CRM does not use an HTTP session in their application. I guess this is true in your case also. You may have used a member variable to store some "invalid" state unintentionally. Member variables are the variables declared by the following syntax:

```
<%! %>
```

For example:

```
<%! Document doc=null; %>
```

Many JSP users misunderstand that they need to use this syntax to declare variables. In fact, you do not need to do that. In most of cases, you do not need a member variable. Member variables are shared by all requests and are initialized only once in the lifetime of the JSP.

Most users need stack variables or method variables. These are created and initialized per request. They are declared as a form of scriptlet as shown in the following example:

```
<% Document doc=null; %>
```

In this case, every request has its own "doc" object, and the doc object is initialized to null per request.

If you do not store an "invalid" state in session or method variables in your JSP, then there may be other reasons that cause this.

Using XMLData From BC4J

Question

I am using XmlData to retrieve data from a BC4J. I Do not use XmlData from a JSP, but from a standalone java application. In the record I target, I have the value 'R & D'.

XmlData returns 'R & D', which is fine for HTTP, but not for my needs. Can XmlData not escape the characters, and just return what's in the database?

Answer

XmlData builds an in-memory DOM, so it must be the XML parser's serialization that's doing this. The only way I know is to do the following:

1. Write your own serializer for the DOM tree that does what you want.
2. Do an identity transform augmented with one template to write that data with disable-output-escaping="yes"

Running XML Parser for Java in JDeveloper 3.0

Question

I have downloaded JDeveloper on my laptop (Windows 95 operating system). I am trying to run a sample XML parser program (SimpleParse.java). This program imports org.w3c.dom.Document class. I have set CLASSPATH in autoexe.bat with correct directory. The program runs on DOS prompt with "java SimpleParse <filename>" command. I am trying to run the same program through JDeveloper but it gives me following error:

```
"identifier org.w3c.dom.Document not found"
```

Am I missing something?

Answer

Make sure to include the Library named:

"Oracle XML Parser 2.0"

is in your project. This library is pre-defined with JDev 3.0and higher and you just need to visit the Project | Properties... and look at the "Paths" tab to see your project's library list.

Click the (Add...) button and pick the above library from the list.

The org.w3c.dom.* interfaces are included in this Jar. They come from the W3C and define the Document Object Model standard API's for working with a tree of XML nodes.

Question 2

Now, if I wish to use the @code as a key, I use

```
<xsl:template match="aTextNode">
    ...
   <xsl:param name="labelCode" select="@code"/>
     <xsl:value-of
 select="document('messages.xml')/messages/msg[@id=$labelCode and
 @lang=$lang]"/>
    ...
   </xsl:template>
```

that works too, but I was wondering if there isn't a way to use the '@code' directly in the 'document()' line?

Answer 2

This is what the current() function is useful for. Rather than:

```
<xsl:param name="labelCode" select="@code"/>
<xsl:value-of
 select="document('messages.xml')/messages/msg[@id=$labelCode and
 @lang=$lang]"/>
```

you can do:

```
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=current()/@code
                      and @lang = $lang]"/>
```

Question 3

And finally, another question: it is - or will it be - possible to retrieve the data stored in messages.xml from the database? How is the 'document()' instruction going to work where listener and servlet will run inside the database?

Answer 3

Sure. By the spec, the XSLT engine should read and cache the document referred to in the document() function. It caches the parsed document based on the string-form of the URI you pass in, so here's how you can achieve a database-based message lookup:

1. CREATE TABLE MESSAGES (lang VARCHAR2(2), code NUMBER, message VARCHAR2

(200));

2. Create an xsql page like "msg.xsql" below:

```
<xsql:query lang="en" xmlns:xsql="urn:oracle-xsql" connection="demo"
            row-element="" rowset-element="">
      select message
        from messages
       where lang = '{@lang}'
         and code = {@code}
</xsql:query>
```

3. Create a stylesheet that uses msg.xsql in the document() function like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
  <html><body>
    In English my name is
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
      </xsl:call-template><br/>
    En espanol mi nombre es
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">es</xsl:with-param>
      </xsl:call-template><br/>
    En fran&#231;ais, je m'appelle
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">fr</xsl:with-param>
      </xsl:call-template><br/>
    In italiano, mi chiamo
      <xsl:call-template name="msg">
         <xsl:with-param name="code">101</xsl:with-param>
         <xsl:with-param name="lang">it</xsl:with-param>
      </xsl:call-template>
  </body></html>
  </xsl:template>
  <xsl:template name="msg">
    <xsl:param name="lang">en</xsl:param>
    <xsl:param name="code"/>
    <xsl:variable name="msgurl"
select="concat('http://xml/msg.xsql?lang=',$lang,'&amp;code=',$code)"/>
    <xsl:value-of select="document($msgurl)/MESSAGE"/>
  </xsl:template>
</xsl:stylesheet>
```

4. Try it out at http://xml/testmessage.xsql

This is great if you want to fetch the message from over the web. Alternatively, you could use the msg.xsql above but include it in your XSQL Page if that makes sense using:

```
<xsql:include-xsql href="msg.xsql?lang={@lang}&amp;code={@code}"/>
```

Or you could write your own custom action handler to use JDBC to fetch the message and include it in the XSQL page yourself.

Moving Complex XML Documents to a Database

Question

I am moving XML documents to an Oracle database. The documents are fairly complex. Can an XML document and the Oracle Developer's Kit (XDK) generate a possible DDL format for how the XML Document should be stored in the database, ideally generating an Object-Relational Structure. Does anyone know of a tool that can do this?

Answer a

The best way may be to use the Class Generator. Use XML SQL Utility if DTD files are not already created. You'll still have to write a mapping program.

Another method is to create views and write stored procedures to update multiple tables. Unfortunately, you'll have to create your tables and views beforehand in either case.

BC4J

Business Components for Java (BC4J) framework provides a general, meta-data-driven solution for mapping E-Commerce XML Messages into and out of the database. BC4J has a technical white paper on its features available at http://otn.oracle.com/products/jdev/info/techwp20/wp.html.

It is a Pure-Java, XML-Based business components framework for making building E-Commerce applications easier. It is a Java framework usable on its own, but also has tight development support built-into JDeveloper 3.0 IDE, available for download from http://otn.oracle.com/software/.

BC4J lets you flexibly map hierarchies of SQL-based "view components" to underlying business components that manage all application behavior (rules and processes) in a uniform way. It also supports *dynamic* functionality, so most of its features can be driven completely off XML metadata.

You can build a layer which flexibly maps any XML Document into and out of the database using this framework. One key benefit is that when XML Documents are put into the system, they automatically can have all the same business rules validated.